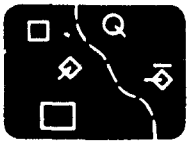
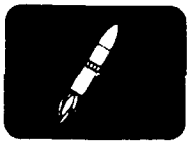
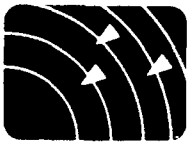
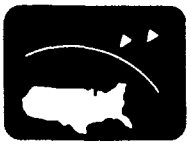
TRW Systems Engineering &
Development Division

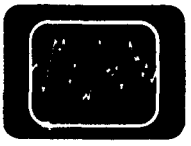
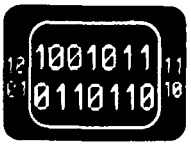
TRW-TS-89-03

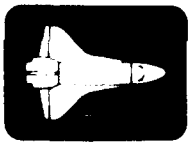
Requirements Analysis for Large Ada Programs: Lessons Learned on CCPDS-R

Charles Grauling

December 1989



TRW Technology Series

TRW Technology Series

TRW Technology Series

TRW Technology Series

TRW Technology Series

TRW Technology Series

TRW Technology Series

TRW Technology Series

hnology Series

Requirements Analysis for Large Ada Programs: Lessons Learned on CCPDS-R

Charles Grauling
TRW Defense Systems Group
Redondo Beach, California

ABSTRACT

This paper summarizes the experiences of the software requirements team on the Command Center Processing and Display System Replacement (CCPDS-R) program. The CCPDS-R program involves the development of large amount of command and control software while using a new approach to the entire software development process. The major challenges that the software requirements team faced include: performance constraints, dealing with uncertainty in a fixed price development environment, and incompatibility of the required deliverable documentation (as specified in DOD-STD-2167) with real project needs.

The techniques included: a small team with good tools, concurrent software development, extensive prototyping and performance modeling, and DID tailoring. The focus of this paper is on the process, techniques and tools that supported the critical front end of a successful large scale Ada development program.

PROJECT BACKGROUND

The Command Center Processing and Display System Replacement (CCPDS-R) program is the missile warning element of the new Integrated Attack Warning/Attack Assessment (ITW&A) system architecture developed by North American Aerospace Defense Command/Air Force Space Command.

The CCPDS-R program was awarded to TRW Defense Systems Group in June 1987. It consists of three separate subsystems of which the first, identified as the Common Subsystem, is 24 months into development and has just completed Critical Design Review (CDR). The Common Subsystem includes approximately 325,000 source lines of Ada with a development schedule of 38 months. Characterized as a highly reliable, real-time distributed system with a sophisticated User Interface and stringent performance requirements, CCPDS-R Ada risks were originally a very serious concern.

The CCPDS-R program is using a new software development process model that features concurrent requirements analysis and software design, early software implementation and capability demonstrations, and incremental integration and formal requirement verification. The overall process is further described in [Royce 1990] and [Springman 1989]. This paper will describe TRW's experiences in the requirement analysis and preliminary design phases of the project.

CCPDS-R SYSTEM and ITS ARCHITECTURE

Figure 1 illustrates the functional interfaces among the CCPDS-R subsystems and their primary external interfaces. There are two deployments of the Common Subsystem (labeled CMAFS and OPCC on the Figure) that consist of identical hardware and software components. The primary function of the Common Subsystems is to process the information received from sensors, generate displays for local consoles, integrate the missile warning information with other manually entered data, and distribute this information to other command centers. The other command centers receive and display this information by using a CCPDS-R Processing and Display Subsystem (PDS). PDS is capable of receiving and displaying direct sensor data and processed data from the Common Subsystem. The SAC command center has additonal processing capability necessary to support its force management and force survival functions. This capability necessitates a high bandwidth link to the Common Subsystems, additional processing resources, and and SAC unique algorithm and display software. The Common Subsystems are capable of mutual backup in the sense that either system can generate the output messages to the other users, PDS, and SAC Subsystem.

The scope of the CCPDS-R program includes the development, production, and installation of the three subsystems as denoted by solid lines on Figure 1. This paper will focus on the requirements analysis process that was used to develop the Software Requirements Specifications (SRSs) for the Common subsystem. The formal software requirements analysis process does not start until the system requirements and ADPE architecture are defined. For the CCPDS-R project, this occurred during the competitive procurement process that

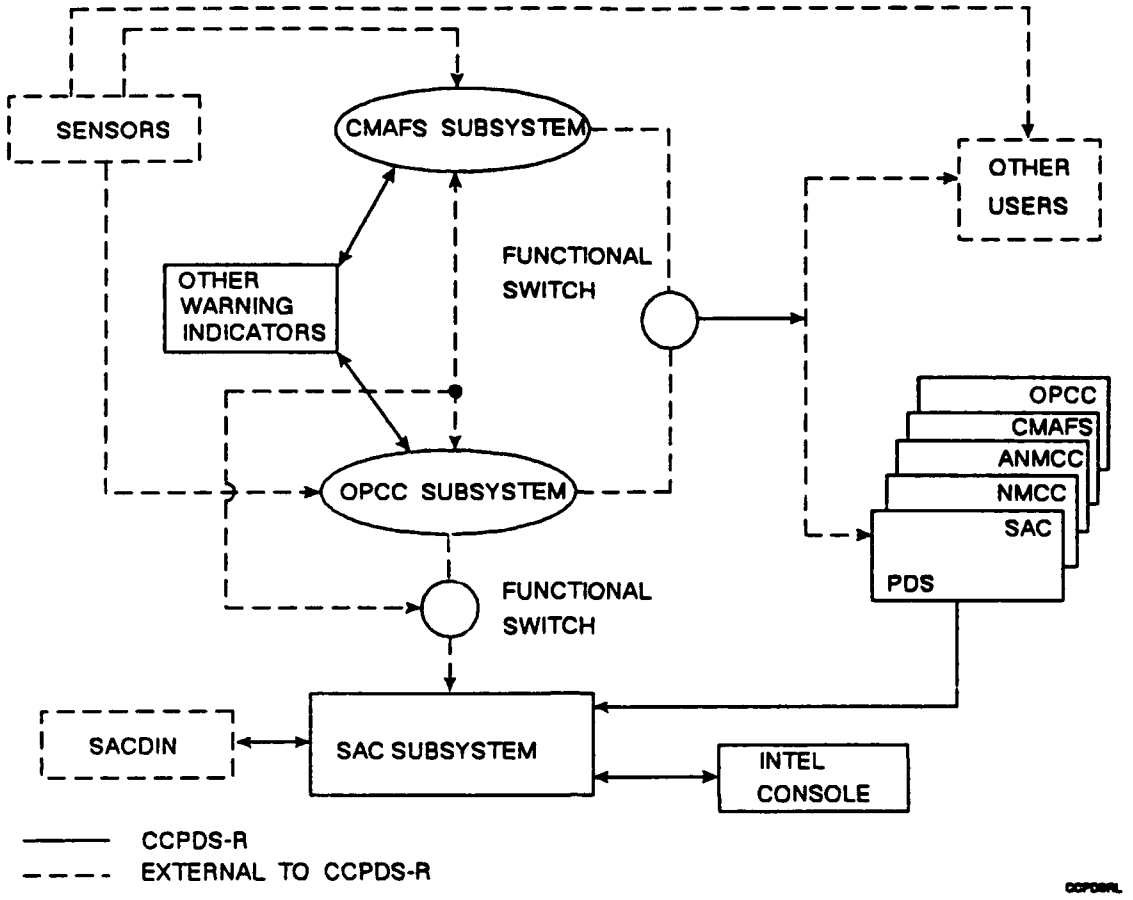


Figure 1: CCPDS-R Sytem Diagram

preceeded the award of the current Full Scale Development contract. The top level system requirements that drove the Common Subsystem's hardware and software architecture prior to award of the current contract include:

- Message processing throughput. The Common Subsystem must be able to execute mission processing algorithms on each incoming sensor message within tight constraints on both elapsed time from message arrival to algorithm results availablity and CPU processing load. These constraints primarily affect algorithm design and the overall software implementation approach.

- Interactive display performance. The information content and requisite responsiveness of the interactive displays was specified. The exact design of the interactive displays was to be determined during the full scale development phase.

- Reliability. CCPDS-R must continuously available and it must always produce valid results. The high availability drove the hardware selection to a distributed architecture with redundant processing resources. A very high mean time between software induced system failures also is required to meet system availability requirements. Explicit data integrity and message accountability requirements were included in the CCPDS-R System Specification to ensure the validity of its outputs. A resulting constraint on the software development process is the necessity to include sufficient testing to verify that both system and software reliablity requirements are indeed satisfied.

- Growth and flexibility. CCPDS-R's expected operational lifetime is more than 15 years. During that time, the entire ITW&A system is expected to evolve. CCPDS-R will have to adapt to changing sensors, message sets, and display requirements.

In general there is a tradeoff between flexibility and performance. The need to properly make that tradeoff affected the requirements analysis and preliminary design process.

The ADPE architecture for the Common Subsystem, in Figure 2, defines the execution environment for the Common Subsystem Computer Software Configuration Items (CSCI). All the hardware components are commercial off the shelf products. The processors are members of compatible family (DEC VAX) running the VAX/VMS operating system. The main mission processing element is a pair of VAX 8800 processors. They handle all external communication traffic (i.e., incoming sensor messages and outgoing warning and assessment messages) via an external communication system (called CSSR). There is a support processor (VAX 8350) that is used for internal test message injection and other non-critical support functions. Each Warning System Operator has a workstation that includes a dedicated processor (VAX 3200). This processor maintains a local copy of the time critical database which is used to update interactive user displays. As a result, the interactive display request responsiveness is quite insensitive to the processing load caused by incoming sensor message traffic. It is primarily sensitive to display complexity in terms of the number of displayable graphic objects contained in the requested display. All processors on the network have access to a common disk pool through a standard DEC software capability called "clustering". The DEC environment also includes DECNET software which provides reliable processor to processor communication services.

The software design team treated the suite of commercial hardware and software products as a single loosely coupled multiprocessor. The developed software that executes on this processor system is organized into six CSCIs.

- Network Architecture Services (NAS). This CSCI performs general purpose processing functions required to provide the loosely coupled multiprocessor ADPE framework for the implementation of the other CCPDS-R CSCIs. It performs intertask communication, network management, and general purpose functions supporting: data recording, network status monitoring, system initialization, and reconfiguration. NAS hides processor and network details from the other CCPDS-R CSCIs. It effectively provides the general purpose "building blocks" needed to implement realtime command and control systems by defining abstract executable objects (called application tasks and processes) and the communication services required to use them. The development of NAS was based on results of a TRW IR&D project whose objective was to use Ada to develop a reusable envirionment for implementing Command and Control systems. Its existence as an early, stable component was crucial to the success of the overall software development process. NAS is completely reusable in the other subsystems. See [Royce 1989] for a more detailed presentation of NAS and its capabilities.

- Common System Services (SSV) CSCI. This CSCI provides top level subsystem control logic and related subsystem support functions including computer system operator interface, indicators and warning (I&W) display generation and data entry, database management and distribution, data recording and reduction, and system moding and control.

- Common Communications (CCO) CSCI. This CSCI handles all external interfaces for the Common Subsystem. It provides the processing required to implement the CSSR upper level communication protocols, validate and reformat incoming messages, and route incoming messages internally within the Common Subsystem. It also performs equivalent processing for all outgoing Common Subsysem generated messages. The primary reason for its status as a separate CSCI was to ensure encapsulation of initially unstable external interface details such as message set definitions and interface protocols.

- Common Mission Processing (CMP) CSCI. This CSCI performs the mission critical algorithms that convert incoming sensor messages into displayable results that support decision making.

- Test and Simulation (TAS) CSCI. This CSCI performs the processing required to support all the test and exercise modes specified in the the CCPDS-R System Specification. The operational CCPDS-R system is continually used in a test and exercise mode in support of crew training and operational readiness. TAS provides the interactive scenario generation and test control, and online message injection during a test or exercise scenario required to support all test and exercise operations.

- Common Display Coordination (DCO) CSCI. This CSCI performs the user interaction and display generation functions for the warning system operator (WSO) position. It also had a headstart due to an IR&D project that had an objective to develop flexible display generation software techniques in Ada. This CSCI's CSCs execute primar-

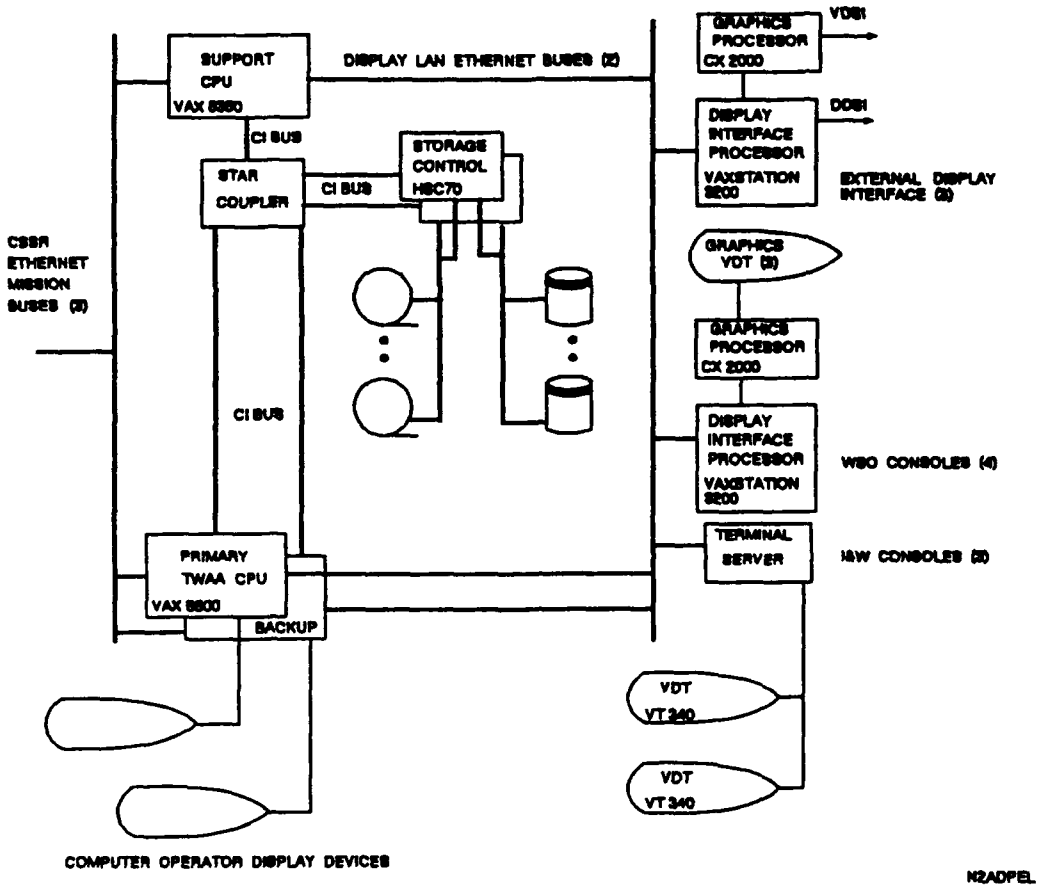


Figure 2: Common Subsystem ADPE Architecture

ily in the VAX 3200 processors contained in the operator workstations.

CCPDS-R's SOFTWARE ENGINEERING METHODOLOGY

The CCPDS-R program is contractually required to use the documentation suite specified in DOD-STD-2167 [DOD 1985]. This includes the development of a Software Requirements Specification (SRS) in accordance with the content and format requirements specified in DOD-STD-2167's Data Item Description (DID) for the SRS. In spite of its title, a DOD-STD-2167 compliant SRS is *not* a pure software requirements document. It certainly contains software requirements, but a DID compliant SRS must contain additional details that cannot be provided without first completing the *design* of the software. CSCI to CSCI interface characteristics which are more detailed than the Ada source code that implements the interface are required. Likewise, required performance characteristics such as storage allocations and timing requirements are by-products of the software design process when multiple CSCIs are executed within a single processor or multiprocessor system.

It was clear at the start of the CCPDS-R full scale development phase that the SRSs would be "living" documents. The contract contained several planned submissions of draft versions that were used by the Government to track the progress of the design process. Initially the SRSs contained the functional requirements necessary to drive the preliminary design process. The SRSs evolved with the software design process and the missing details were filled in by CDR as planned. The process that was used to produce the Common Subsystem SRSs consisted of the concurrent requirement analysis and software design analysis activities illustrated in Figure 3.

The primary inputs to the CCPDS-R software en-

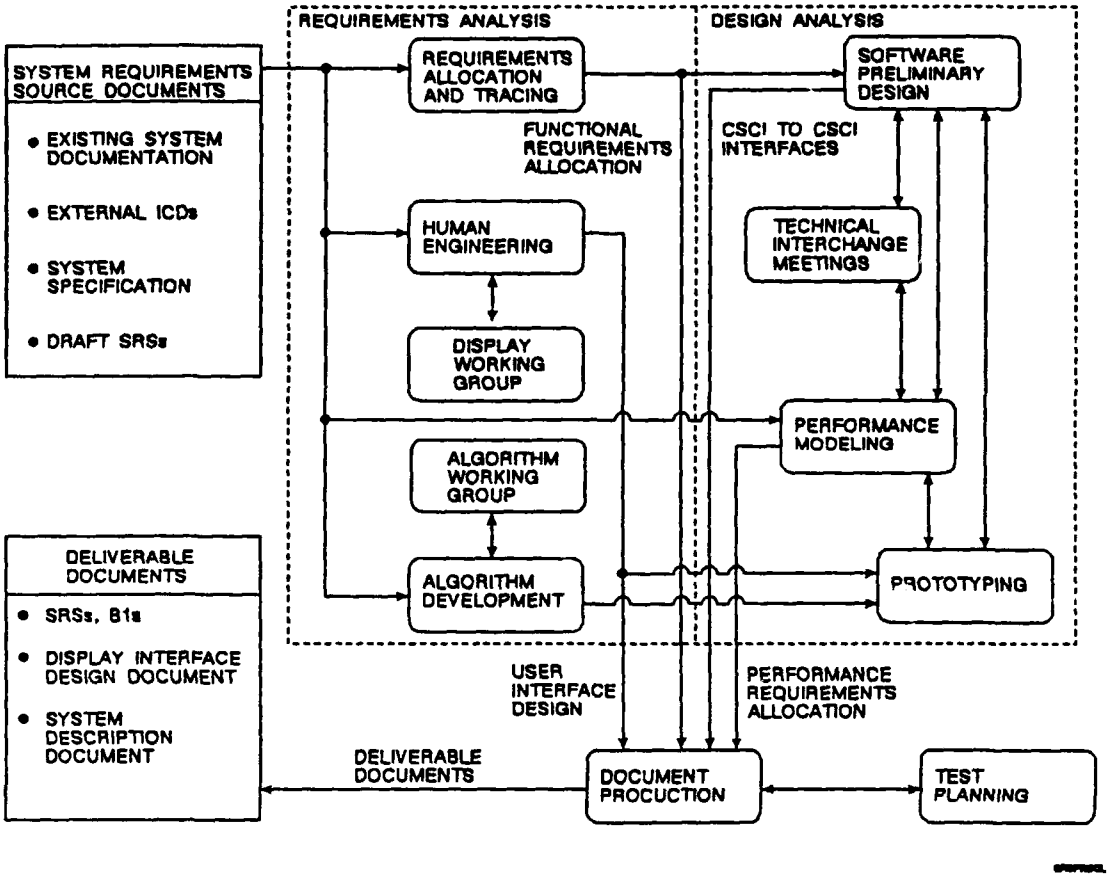


Figure 3: CCPDS-R Software Engineering Process

gineering process are the documents indicated on Figure 3 that existed at award of contract. The word "Replacement" in CCPDS-R refers to the fact that CCPDS-R is a replacement of an existing system. As such, CCPDS-R has a system level requirement to generate messages that are identical to those generated by the current system it that will replace. Hence, existing system specifications and Interface Control Documents (ICDs) are source documents for the CCPDS-R system requirements analysis process. These documents in conjunction with a stable CCPDS-R System Specification and draft SRSs that developed during the competitive design phase adequately defined the system requirements baseline for the software design process that was about to start.

Requirements Allocation. The first step in the software engineering process was formal requirements allocation and traceability database generation. The source documents were analysed sentence by sentence. Each sentence that contained requirements (i.e., included the keyword "shall") was tagged with a unique identifier and entered into a database. Each requirement was then allocated to one or more Configuration Items (CI) and the allocation relationships were also entered in the database. In this context, "a requirement is allocated to a CI" means that the SRS or CI development specification associated with the CI must contain requirements that trace back to the allocated source requirement. We developed automated consistency check procedures and reports based on this traceability database. The first reports were simple sorts by CI. They were used by SRS authors to ensure that their SRSs were complete with respect to requirement coverage. Formal requirement traceability was included in the process from the start with the engineers responsible for generating the traceability relationship data. We had a full time traceability toolsmith and enforcer to help ensure that all requirements were properly traced.

With the functional requirements, performance characteristics, and hardware architecture already de-

fined and contractually constrained, the only significant remaining degrees of freedom in the system design were in the specific design of the message processing algorithms and the warning system operator interface. The program plan included Joint Government/Contractor Working Groups to handle these critical system design issues. The Working Groups' tasks were to design algorithms and displays that meet the System Specification's functional requirements within the performance budgets imposed by the contract's hardware baseline. The working groups ran in parallel in support of the Human Engineering and Algorithm Development activities indicated on Figure 3. The resulting display and algorithm designs were input to the prototyping activity. In addition to the display designs, the human engineering activity also produced user interface characteristics requirements that were to be included in the SRSs for CSCIs that implement user interface software. These requirements were derived from military standard human engineering guidance documents.

The design analysis part of the process consists of three related activities: software preliminary design, prototyping, and performance modeling. Software preliminary design on CCPDS-R could start earlier than usual because the requirements independent part (NAS) had already been specified and prototyped prior to the award of contract. The NAS capabilities needed to support the top level design were available and their interfaces were stable. Likewise, a baseline software architecture already existed as a by-product of the competitive design phase. The inherent flexibility provided by the NAS components and message based software allowed us to perform top down design and integration early in the process without fear of costly code breakage down stream. The software preliminary design was supported by benchmarking and prototyping of critical algorithms. In addition to the mission processing algorithms, we prototyped the software that performs inter task communication, display generation, and time critical database distribution. The prototyping provided insight about performance bottlenecks and valuable experience about the processor resource demands of the commercial software components.

Performance Modeling. We supported the design process with independent performance modeling. The performance model uses discrete system simulation to maintain a prediction of processing resource usage and system responsiveness. It uses a mixture of initial estimates, software designer supplied line of code counts, and empirically derived performance data obtained from the prototyping activity. The fidelity of the model was continually upgraded as design details emerged out of the preliminary design activity and empirical performance data was accumulated by the prototyping activity. A more detailed description of this activity is provided by [Viceroy 1990]. The primary effect of the software design analysis activities on the SRS development process was to substantiate the design details that go into the SRSs and aid in focusing engineering resources on the difficult or critical design issues. Another useful feature of this approach is that it allowed us to rationally design to performance constraints. The first versions of the top level software architecture were optimized to support integration. They had many small tasks with simple interfaces. Bottlenecks and improperly functioning components were easy to detect and isolate in this environment. Once the system was integrated and all the obvious blunders corrected, the process of squeezing the software into the performance constraints involved consolidating tasks to reduce communication costs and operating system overhead. This process tends to result in a system that is more computer resource efficient at a cost of being harder to understand and maintain. The tradeoff between efficiency and other less measurable software quality attributes stops when the software fits into the performance constraints imposed by the system specification. This approach effectively settles in on an optimal solution because it minimizes the amount of unmeasurable software design quality attributes that are expended to accomplish the measurable CPU efficiency requirements.

Document Production. It may seem unconventional to include document production as an engineering activity as indicated on Figure 3. However, the contractual data requirements were sufficiently difficult to manage that we applied engineering talent ensure that they could be satisfied. There were five planned submissions of the Common Subsystem SRSs including one with the proposal, one prior to each of the major reviews: System Requirements Review (SRR), Preliminary Design Review (PDR), Critical Design Review (CDR), and a final authentication version following the Critical Design Review. In addition, the CCPDS-R contract requires numerous other documents such as plans and procedures, system description document, and DOD-STD-2167 software design documentation. Many of these documents are required to contain common information, or details that could be extracted mechanically out of other documents. We used a public domain text formatting program (LaTeX) [Lamport 1985] as a basis for building an automated document production and maintenance system. We substantially reduced the manpower required to maintain consistency across CCPDS-R's large collection of related documents by surrounding LaTeX with command procedures and file management conventions to develop this system. LaTeX is best described as a document compiler. Our document authors generate source code written in the LaTeX programming language. We

then build deliverable documents by compiling and linking modular documentation components. We also apply familiar software design and database management principles to the document production process. For example, we apply the principle of single point of control by requiring that text which must appear in more than one document (such as common general purpose descriptive material or interface characteristics) must be contained in individual LaTeX source files and imported into each using document. In this environment, document production is more than just turning the paperwork crank. The automated documentation techniques are an essential component of the discipline that we use to perform System Engineering.

The key to allowing the SRSs to evolve without creating chaos is having the ability to produce and maintain the documents efficiently while keeping track of their relationship to the contractually binding System Specification. The documents must be available throughout the requirements analysis and preliminary design phase even though they are evolving in response to system requirements interpretation and preliminary design results. Our production and maintenance methods permitted this. We used a combination of commercially available tools (document formatter and structured analysis tool) in conjunction with locally developed utilities to provide an effective integrated document production and requirements traceability environment. The document production capability and requirements traceability system are two separate systems that are linked by shared files.

The document production system is illustrated in Figure 4. Document authors create LaTeX source code using the the text editor available on our host computer facility. This source code defines abstract text objects such as paragraphs, lists, floating figures and tables, cross reference labels, etc. LaTeX also has fairly powerful but difficult to use graphics capability that involves specifing a picture in Cartesian coordinates. This capability was used only by hard core computer programmer types. Others use a commercially available PC-based graphics editor. After editing a source file, the user runs the LaTeX program. LaTeX calculates how the text objects will look when they are typeset onto paper. It builds a device independent representation of each output page which is stores on a disk file (called a "DVI Files"). In addition, LaTeX builds auxilliary files which contain other information which is used to build front matter such as the table of contents and list of figures. Drawing files created on the PC are down loaded to the host computer and available for printing on the host computer's laser printer or inclusion into a LaTeX document. Printed output is generated by running a printer specific driver program, QDrive, that was provided by the laser printer vendor. QDrive reads a LaTeX DVI file and converts the device independent representation into the appropriate printer commands to produce the final printable file. By embedding the appropriate LaTeX commands in the source file, it is possible to instruct QDrive to import a graphics file and scale it into a predetermined space on a page. This paper was produced using the CCPDS-R document production system.

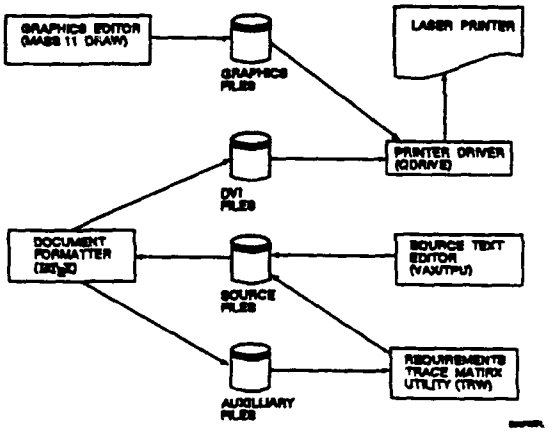


Figure 4: Document Preparation System

The LaTeX language is extensible via a macro facility. We used this extensibility feature to create many new abstract text objects to enhance the SRS creation and production process. One such extension, the "requirement" abstract text object, provided the link to the traceability system. These "requirement" objects expand into the printed text as numbered "shall" (i.e., they look like this – shall[1]). The numbers are automatically assigned by LaTeX to ensure that all requirements were uniquely labeled. In addition, LaTeX captures other information from the document context and writes it out to an auxilliary file. This other information includes the automatically paragraph number and title, and author entered source reference traceability information, and verification cross reference matrix information. We then wrote a program to read this auxilliary file and generate the LaTeX source code which builds the requirement traceability and verification cross reference matrices that go in each SRS. This technique saves work and produces correct consistent results. We also use the same auxilliary file to populate the requirements traceability database.

Requirements Tracing. Figure 5 illustrates major components of the CCPDS-R requirements traceability sytem. The process started with manual requirements allocation. To prepare for this step, we dissected the system specification on a sentence by sentence basis

and entered each sentence and its address (i.e., paragraph number concatenated with its sentence number concatenated with its list item identifier if the sentence contained a nested list) into a relational database. This allowed us to use address strings as unique identifiers for individual system requirements. The manual requirement allocation step was accomplished by conducting a marathon meeting with all the CSCI authors to determine which CSCIs would be "responsible" for each of the system specification's verifiable requirements. The decisions made in this way were entered into the database and we could generate a report for each CSCI author that contained the CSCI's subset of requirements to be traced.

Properly armed with their allocated requirements report, the SRS authors could then incorporate the source reference traceability information in the SRSs as they were created. This is the most natural way to capture source reference traceability information. The SRS author has the best knowledge of which system specification requirement(s) are the source of each SRS requirement during the process of the SRS LaTeX source code. Capturing that information at the time when the SRSs initially are created is the least painful way to do it. This approach was initially devised a way of automating production of the source reference tracability table. A minor modification to the requirements trace matrix utility shown in Figure 4 turned it into a Traceability Database Populator shown on Figure 5. This allowed us to automatically capture the source reference traceability information from the SRSs. It was easy to implement automated analysis of the consistency of the traced requirements against the allocated requirements. Error reports generated by the traceability analysis process allowed us to uncover a variety of errors early in the process and provided confidence that we had indeed covered all system specification requirements.

The validity of the process is still based entirely on engineering judgement. A computer program cannot assess the "correctness" of the either downward allocation or upward traceability. However, the computer's outstanding ability manage a large mass of traceability data without getting bored or making mistakes allowed us to do a more thorough and accurate job.

PROBLEMS AND SOLUTIONS

Varying role of the SRS. The primary purpose of the SRSs is to provide the software requirements baseline. They are initially used to resolve ambiguities in the system requirements and/or clarify their interpretation. Next they are used to drive the software design process. Ultimately the SRSs become the basis for the software formal verification and acceptance. The participants in the SRS development process include: System Engineers, Software Engineers, Testers, and Managers from both the Contractor and Government communities. Throughout the SRS evolution, period, they tend to be treated as contracts among the various factions involved in their creation and use. As such, they become a focal point for conflicts and their resolution. As the system design process goes through different phases, the different factions become more or less active in the process. The best way to illustrate this point is to chronicle what happened on CCPDS-R with each of the released SRS versions.

1. SSR Version. The first draft SRSs focused on defining the scope of processing requirements by attempting to resolve open system specification interpretation issues. Their primary function was tutorial. They were used by the software developers as design guidance. They also were used to communicate system design progress to the Government review team. They contained information that was derived primarily from the system specification and numerous Technical Interchange Meetings (TIMs) with end users. The focus of the TIMs was primarily database content and performance issues.

2. PDR Version. The primary feedback from the review of the SSR version came from the Government review team and the software development team. In general, the first version of the SRSs contained too much design detail for the software developers and not enough for the Government review team. This was caused by the differences in their respective responsibilities. The developers are responsible for find a design solution that meets the functional requirements within a performance budget. They need design flexibility. Whenever the SRS specifies how to do something rather than what must be done, implementation flexibility is lost and it becomes harder to find a performance compliant solution. The Government review team has a different responsibility. They are tasked with determining if the emerging design will work. Early in the process, when design documentation and demonstrable products were not yet available, the Government team exhibited an enormous appetite for detail. Armed with a DID that can be interpreted as requiring considerable design detail (such as CSCI to CSCI interface designs, error processing details, internal timing and sizing allocations), they exerted considerable pressure on us to augment the SRSs with additional and more detailed requirements.

   The most exaggerated example of this phenomenon occurred in the human interface area. The DCO developers wanted the SRS to specify required general

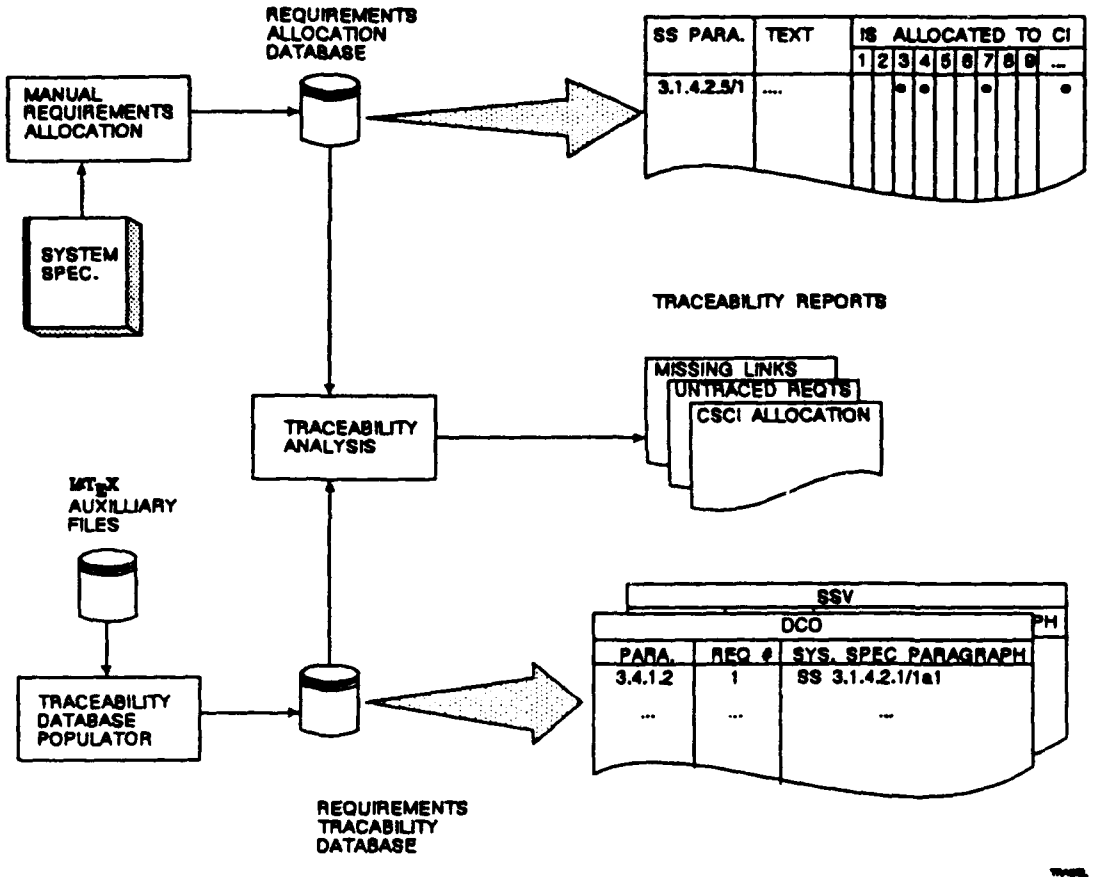


Figure 5: CCPDS-R Requirements Traceability System

purpose capbilities that would provide a sufficiently flexible environment for implementing a To Be Determined design for displays and user interaction. The Government review team felt that the user interface was an external interface. Therefore, function key hits, graphics picks, and user keyboard inputs were external inputs and it was necessary for the SRS to specify the exact processing required and resulting outputs for each type of user input. We resolved this conflict through a number of compromises. We generated an additional document, the Display Interface Design Document (DIDD), that contained the desired clarifying details. We added more requirements in the SRS which identified specific required displays by reference to the DIDD. This allowed us to stabilize the SRS while letting the detailed design and implementation proceed. Interactive display responsiveness is one of the most difficult requirements to design to because it is very sensitive to the details of the display layout and their information content. After the PDR version was issued, the software developers pressed on generating operational code, and the incremental testing and verification process was started [Springman 1989].

3. CDR Version By CDR, most of the operational system was built, informally tested and intergrated. The CDR operational demonstration was close enough to meeting performance requirement to indicate that performance requirements will be satisfied through tuning and minor enhancements. The emphasis for the CDR version of the SRSs was to modify them to enhance their usability as a basis for formal selloff of the system. This involved removing details that were now a source of confusion and conflict among the testers and their reviewers.

Incompatible perceptions and uses of the SRS guarantee that someone will always be unhappy with the SRS and its contents. In real-world system de-

velopment, requirements and design approaches evolve continuously. In general, the CCPDS-R SRSs tracked that evolution. Initially they were design oriented with increasing implementation constraining detail. Toward the end of the process, when the design had matured and the SRS role was to be the tester's contract, implementation detail was removed and replaced specific requirement definition language which was more suitable for formal verification. The SRS development methodology has to support requirement evolution and the changing role of the SRS within the overall development process. Our automated approach to document production and maintenance provided this support.

DI-MCCR-80025 Interpretation. The DOD-STD-2167 Data Item Description (DI-MCCR-80025) that specifies the format and content of the SRS. DI-MCCR-80025 is most appropriate for an embedded computer type of application where there is one CSCI running in one processor. CCPDS-R consists of multiple CSCIs coexisting on a loosely coupled multiprocessor system. The differences between the two environments cause some DI-MCCR-80025 requirements to be inappropriate. A good example of this is DI-MCCR-80025's requirements to specify CSCI to CSCI interfaces in terms of low level details. In the CCPDS-R ADPE environment, CSCI to CSCI interfaces are essentially internal interfaces. They are implemented by using packages of Ada record type specifications that are imported by components in the interfacing CSCIs. Most of the implementation details that DI-MCCR-80025 requires are irrelevant for Ada software development. Of course, the information content of the interface in SRSs is important. We needed a DID that allowed the use of abstract data types in interfaces. There were many other examples of DI-MCCR-80025 specifics that were inappropriate for large scale Ada developments. We negotiated with the Government review team to develop a mutually agreeable "interpretation" of the DID requirements that made sense for CCPDS-R. DI-MCCR-80025 has since been superseded by DOD-STD-2167A [DOD 1988]. The new DID is more appropriate for the CCPDS-R environment. The modifications that we put in our internal version of the DID were remarkably consistent with the new DID.

CSCI Partition. The top level decomposition of CCPDS-R into CSCIs done from the software development perspective. It achieved effective encapsulation of the important sytem attributes such as external interfaces (including protocols, message sets and their information content), mission algorithms, user interface characteristics, and top level system control. It forced early defintion and configuration control on a few key internal interfaces and data structures which subsequently enabled the developers to work their design problems in relative isolation. The developers were very comfortable with it. The smoothness and efficiency of the low level testing and integration verified that the basic approach was sound.

This approach was not optimal from the formal testing and requirement verification perspective. The concept of treating subsystem internal CSCI to CSCI interfaces as external interfaces in the same manner as real external interfaces (such as the incoming sensor message sets) simply didn't work. The information content requirements contained in the SRSs mapped into Ada record type specifications in the implementation. A variety of mechanisms for passing objects of a given type among CSCIs were available to implement interfaces within the CCPDS-R software architecture. The choice of CSCI to CSCI information interchange mechanism is an implementation decision based on performance tradeoffs. It doesn't belong in the SRS because it might need to change any time prior to formal system test. The formal testing community uses the SRS as a basis for test planning and test case design. It can't determine how to explicitly verify interface requirements if the SRSs don't contain the missing implementation detail.

We dealt with this issue by working out a common sense arrangement with the Government review team to effectively conduct a proper requirements verification process without arbitrarily constraining the design. Rather than explicitly verifying the internal CSCI to CSCI interfaces, we defined a new verification method called "Implicit" verification. A requirement may be verified implicitly if there is a test case that verifies some other requirement(s) which could not have worked properly had the implicit requirement not also been satisfied. This situation happens frequently in CCPDS-R with its functional CSCI partition and message based design approach. For example, a sensor message enters the system gets validated and reformatted by one CSCI that sends it to another CSCI for processing to generate results that are displayable by another CSCI. The test case that verifies proper operation of this processing thread implicitly verifies that the pertinent internal CSCI to CSCI interfaces are working correctly. We could now finalize the SRS and generate valid test plans and procedures without arbitrarily constraining the implementation of CSCI to CSCI interfaces.

In retrospect, we effectively treated the entire collection of CSCIs as if they were a single CSCI for requirement verification purposes. If we had had a single CSCI for the entire Common Subsystem with the same first level of decomposition into functions as the current CSCI set that we used, we would have generated less paperwork and no loss in visibility or design control.

We are about to restart the software requirements

analysis and preliminary design for the third CCPDS-R subsytem (SAC). Based on Common Subsystem lessons learned, the SAC Subsystem will consist a single developed CSCI that executes under the VAX/VMS and NAS.

SUMMARY

The software engineering and preliminary design process was successful on CCPDS-R for the following reasons:

- We had a small team consisting of five senior engineers built the SRSs for all three subsystems including document production. In the process, we developed new ways of generating and managing the software requirements that enhanced both productivity and the quality of the products.
- Concurrent software requirements analysis and design worked out better than we had expected. The detailed performance data that we got out of the early integration and prototyping helped us know where to draw the line on algorithm and display complexity. Moreover, the early implementation allowed us to demonstrate operational capabilities to the CCPDS-R user community. This provided us with vaulable user feedback. It also gave the Government a much better understanding of the state of the design than is possible with the pile of documents and briefing charts that is normally available at design reviews.
- We tailored the DID so that it made sense for CCPDS-R. Some tailoring was specifically included in the contract. Additional DI-MCCR-80025 interpretation agreements during the system design activity prevented the need to include worthless information ir the documents solely to satisfy the DID.
- The CCPDS-R System Specification is an excellent document. We had only minor squabbles over its interpretation. It provided detail (approximately 2000 verifiable requirements) without imposing arbitrary design constraints. Concurrent software requirement analysis and software development could not have worked without the System Specification's stable requirements base.

ACKNOWLEDGEMENTS

The success of the CCPDS-R project to date is due to multiple contributions including the TRW System Engineering and Development Division's Management commitment to follow through on a risky new technology insertion with strong support, the United States Air Force Electronics System Division's courage and foresight to allow us to use this new technology on a critical program, and the entire CCPDS-R Software and System Engineering team. Explicit acknowledgements are due to Don Andres, Joan Bebb, Chase Dane, Ron Louie, Elliott Henry, Tom Herman, Steve Patay, Walker Royce, Patti Shishido, Mike Springman, and Darrell Yocom whose day to day involvement and commitment have made this process a success.

BIOGRAPHY

Charles Grauling is the Software Chief Engineer on the CCPDS-R Project. He received his BS in Electrical Engineering from Cornell University in 1966, MS in Electrical Engineering from the Massachusetts Institute of Technology in 1968, and MS in Computer Science from the University of Southern California in 1972. He has been responsible for software requirements analysis and architecture design on the CCPDS-R project since 1984.

REFERENCES

[DOD 1985] DOD-STD-2167, Military Standard Defense System Software Development, 4 June 1985.

[DOD 1988] DOD-STD-2167A Military Standard Defense System Software Development, 29 February 1988.

[Lamport 1985] Lamport, Leslie, "LaTeX: A Document Preparation System", *Addison-Wesley Publishing Company, 1985*

[Royce 1989] Royce, W. E., "Reliable, Reusable Ada Components For Constructing Large, Distributed Multi-task Networks: Network Architecture Services (NAS)", *TRI-Ada Proceedings, Pittsburgh, October 1989.*

[Royce 1990] Royce, W. E., "TRW's Process Model for Incremental Development of Large Software Systems", Submitted to *12th International Conference on Engineering*, Nice, France 1990.

[Springman 1989] Springman, M. C., "Incremental Software Test Methodology for a Major Government Ada Project", *TRI-Ada Proceedings, Pittsburgh, October 1989.*

[Viceroy 1990] Viceroy, J. A., "System Performance Analysis with an Ada Process Model Development", Submitted to *12th International Conference on Engineering*, Nice, France 1990.